

Figure 1

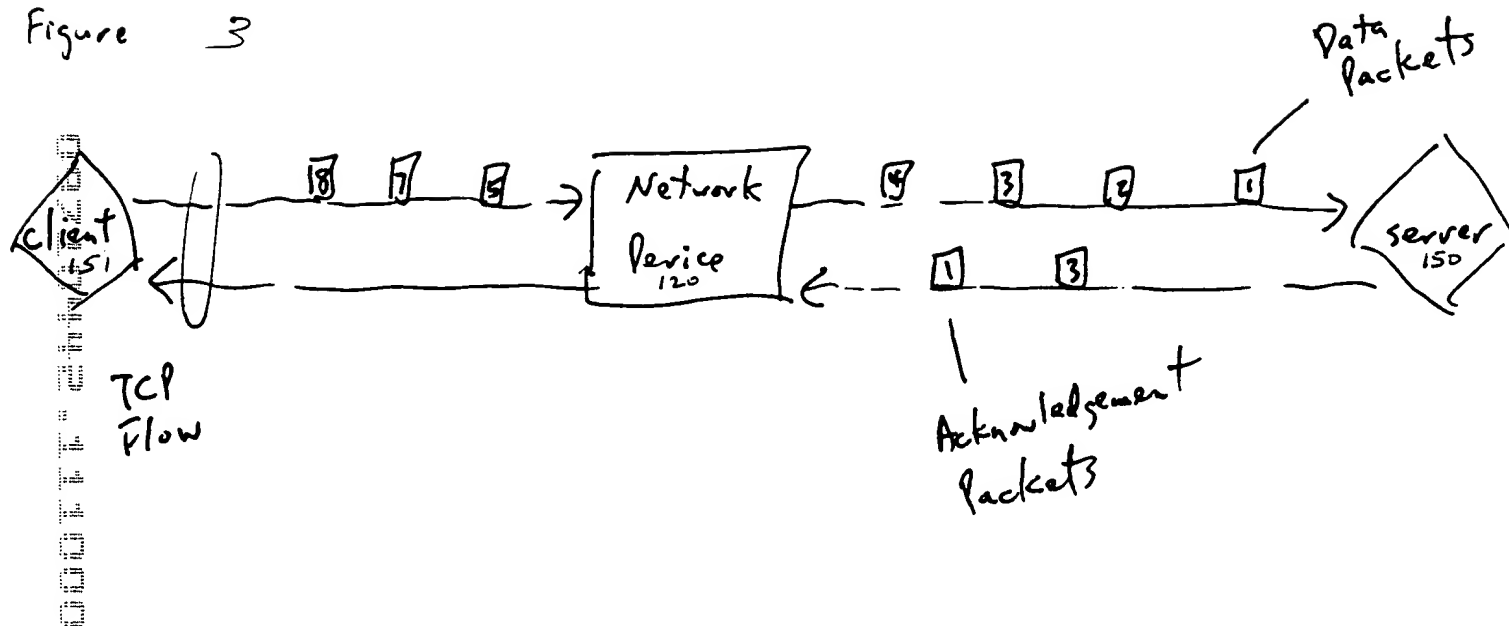
[illegible]

The diagram illustrates the correlation between End-to-End Response Time and Quality of Experience. It shows a network architecture with the following components and delays:

- Customer WAN (260):** The network segment between the Customer and the Customer Data Center LAN.
- Customer Data Center LAN (242):** The local network segment within the Customer Data Center.
- Application Demarc (201):** The point where the Customer's network meets the Service Provider's network.
- Network Service (202):** The network segment provided by the Service Provider.
- End-to-End Response Time - "Quality of Experience" Correlation (210):** The total time taken for a request to travel from the Customer to the Service Provider and back.
- Inbound Customer Network Delay (241):** The delay experienced by the request as it travels from the Customer to the Application Demarc.
- Outbound Customer Network Delay (203):** The delay experienced by the response as it travels from the Application Demarc back to the Customer.
- Inbound Provider Network Delay:** The delay experienced by the request as it travels from the Application Demarc to the Network Service.
- Outbound Provider Network Delay:** The delay experienced by the response as it travels from the Network Service back to the Application Demarc.
- Host Latency:** The delay experienced by the request and response as they wait at the Application Demarc.

Figure 2

Figure 3

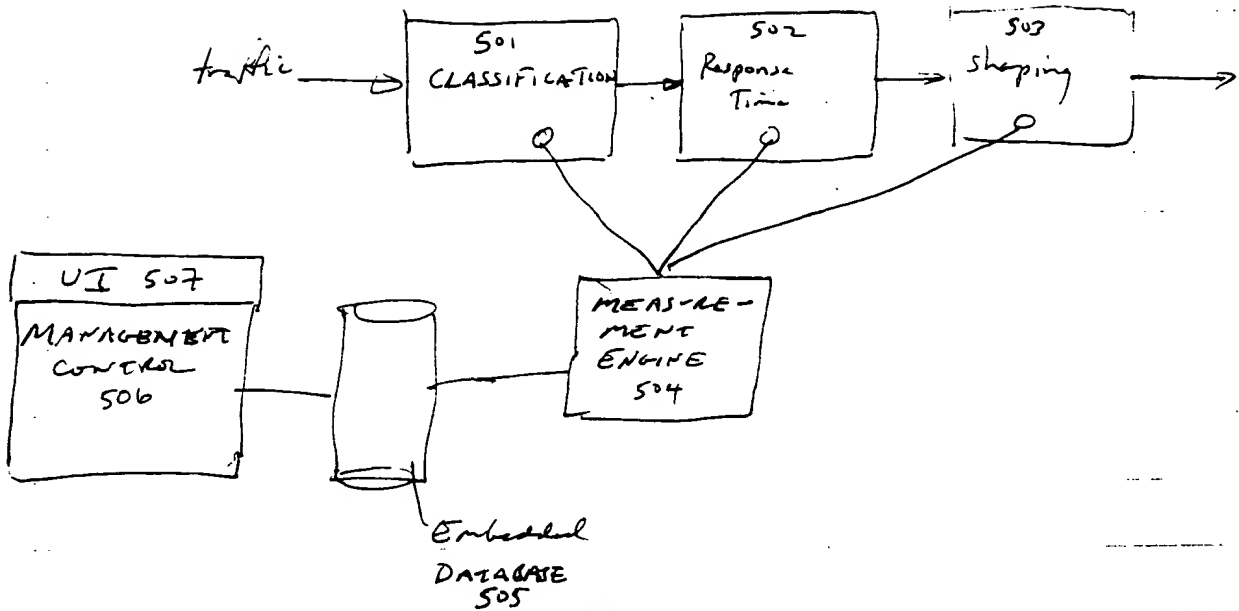


03/22/00  
14:20:24

SCANNED, #  
drixoral.c

Figure 4

```
1  /*.....
2  *
3  * Module Name: drixoral.c
4  *
5  * Abstract: Congestion Index Measurement
6  * Author: Guy Riddle
7  * Created: 991102
8  *
9  * Revision: 1.1.4.6 $Name: $
10 * Date: 2000/01/22 22:20:24 $ Copyright Packeteer, Inc. 1999
11 *.....
12 */
13
14 #include <unistd.h>
15 #include <netcom.h>
16 #include <malloc.h>
17 #include <../tcp.h>
18
19 #if _PSAPPV
20
21 #define MY_DEBUG_LEVEL gTCPRTNDebugLevel
22 extern int gTCPRTNDebugLevel;
23
24 #define metClassCongestionAccum(tc, msec) \
25 metClassPktExchMsecAccum(tc, msec).metClassPktExchSamplesAccum(tc, 1)
26
27
28
29 #define DRIX_SEQ_SPOTS 7 /* chosen to avoid kmalloc roundup wastage */
30
31 typedef struct {
32     SEQ lastseq;
33     SEQ lastack;
34     SEQ seq(DRIX_SEQ_SPOTS);
35     TICK time(DRIX_SEQ_SPOTS);
36 } CongestionMeasurement, *CongestionMeasurementPtr;
37
38
39 void
40 drixoralMeasureCongestion(
41     TCB_INFO_PTR info
42 ) {
43     TCB_PTR tcb;
44     ArtMeasurementPtr am;
45     CongestionMeasurementPtr drix;
46     TCLASS_PTR tc;
47     DIRECTION dir;
48     SEQ seq;
49     msec;
50     int i, leave;
51
52     whereStr("oodMeasureCongestion");
53
54     tcb = info->tcb;
55     am = tcb->artData;
56     drix = info->drix;
57
58     blurt3("seq %d ack %d flags %X", info->header.seq, info->header.ack, info->header.flags)
59     ;
60     blurt4("dir %d len %d end %d tcb %X", dir, info->header.dataLen, info->header.seq + info
61     ->header.dataLen, tcb);
62     if((info->header.dataLen > 0) {
63         if((am->drixoral[dir])
64             am->drixoral[dir] = kmalloc(sizeof(CongestionMeasurement), M_DRIXORAL);
65         if((drix = am->drixoral[dir])) {
66             seq = info->header.seq + info->header.dataLen;
67             if((info->header.flags & (TCP_F_SYN|TCP_F_FIN))
68                 seq++;
69             if((drix->seq, drix->lastseq) || !drix->lastseq) {
70                 .or(i = 0; i < DRIX_SEQ_SPOTS; i++)
71                     if(!drix->seq[i])
72                         break;
73             }
74             if(i < DRIX_SEQ_SPOTS) {
75                 drix->seq[i] = seq;
76                 drix->time[i] = info->bcb->tick_ticks;
77             } else
78                 info2("req %d spot overflow tcb %X", seq, tcb);
79             drix->lastseq = seq;
80         }
81     )
82     )
83     )
84
85     dir = OTHER_DIRECTION(info->dir);
86
87     if(IS_FLAG_SET(info->header.flags, TCP_F_ACK) && (drix = am->drixoral[dir]))
88         && (SEQ_GT(info->header.ack, drix->lastack) || !drix->lastack) {
89             leave = 0;
90             for(i = 0; i < DRIX_SEQ_SPOTS; i++)
91                 if(!drix->seq[i])
92                     continue;
93             else if(SEQ_EQ(info->header.ack, drix->seq[i])) {
94                 if(info->header.dataLen == 0) {
95                     msec = TICKS_TO_MSECS_ROUNDED(info->bcb->tick_ticks - drix->time[i]);
96                     if((tc = tclassIdToTclassFast(tcb->halfConnId, gear.classId)) {
97                         metClassCongestionAccum(tc, msec);
98                     }
99                     blurt4("sample %d msec %d class %s dir %d", i, msec, tc->name, dir);
100                     leave
101                     }
102                     attnl("no class for TCB %X", tcb);
103                 }
104                 drix->seq[i] = 0;
105                 else if(SEQ_GT(info->header.ack, drix->seq[i]))
106                     drix->seq[i] = 0;
107                 else
108                     leave++;
109             if(leave)
110                 drix->lastack = info->header.ack;
111             else {
112                 kfree(drix);
113                 am->drixoral[dir] = 0;
114             }
115         }
116         void
117         drixoralCleanup(
118             TCB_PTR tcb
119         ) {
120             ArtMeasurementPtr am;
121             CongestionMeasurementPtr drix;
122             am = tcb->artData;
123             if((drix = am->drixoral[DIR_INBOUND])) {
124                 kfree(drix);
125                 am->drixoral[DIR_INBOUND] = 0;
126             }
127             if((drix = am->drixoral[DIR_OUTBOUND])) {
128                 kfree(drix);
129                 am->drixoral[DIR_OUTBOUND] = 0;
130             }
131         }
132         if((drix = am->drixoral[DIR_INBOUND])) {
133             kfree(drix);
134             am->drixoral[DIR_INBOUND] = 0;
135         }
136         if((drix = am->drixoral[DIR_OUTBOUND])) {
137             kfree(drix);
138             am->drixoral[DIR_OUTBOUND] = 0;
139         }
140         if(!drix)
141             break;
142     }
143 }
```



✓  
figure 5